様式６

申請先学部　工　学部　採択番号　No.５

２０１９年度大阪大学未来基金【住野勇財団】学部学生による自主研究奨励事業研究成果報告書

ふりがな 氏名	ひらお　かずよし 平尾　和睦	学部 学科	工学部 応用理工学科	学年	３年
ふりがな 共同 研究者氏名	かわかみ　きょうへい 川上　恭平	学部 学科	工学部 応用理工学科	学年	３年
	いずぶち　れお 泉渕　礼於		工学部 電子情報工学科		３年
					年

アドバイザー教員 氏名	大須賀　公一	所属	工学研究科 機械工学専攻
研究課題名	Depth カメラによる外界認識と NDT マッチングによる自己位置推定		
研究成果の概要	研究目的、研究計画、研究方法、研究経過、研究成果等について記述すること。必要に応じて用紙を追加してもよい。（先行する研究を引用する場合は、「阪大生のためのアカデミックライティング入門」に従い、盗作剽窃にならないように引用部分を明示し文末に参考文献リストをつけること。）		

本研究の目的

近年のロボット工学の発達は目まぐるしく、今まで人にしかできないと思われてきたことが次々と可能になってきている。特に注目を集めているものの一つとして車の自動運転があげられる。近い将来実用化されるであろうが、この技術はなにも車に限った話ではないはずである。より小さく小回りの利くロボットに搭載できれば建物内での物資運搬など様々なことに活用できると考えられる。しかしこれには大きな課題がある。自動運転車は高価なセンサーと大規模な計算機を用いて稼働しているためそれをそのまま利用することはできない。では、安価なセンサーと小型の計算機ではどれほどのことができるのであろうか。それを検証するべく、自動運転に不可欠な技術である深度センサーによる自己位置推定を行う。

実験装置

- Intel 社製「BOXNUC8I5BEH」

  この製品はストレージとメモリを別途購入することで計算機として利用できる。またストレージは SSD128GB、メモリは 16GB を採用した。

  大きさは幅 117mm 、奥行き 112mm 、高さ 51(H)mm でこのサイズでは最高レベルの計算能力を持つ。19V を背面のジャックに印加することで駆動するため、市販の 22.2V の Li-Po バッテリーを降圧することで駆動すると考えられる。（実際は電源のことを失念していたため付属の AC アダプターを利用した。）

図１　Intel NUC「BOXNUC8I5BEH」

- Intel 社製「RealSense D435i」

　距離が測定できる特殊なカメラであり、本研究におけるセンサーである。通常の自動運転の深度センサーはレーザーセンサーである LiDAR を用いるが、本製品は赤外線を利用するため値段が安い分精度が悪い。また赤外線は太陽光によって乱されるため屋外は苦手と言われる。本製品は内部に IMU も搭載しており、加速度、角速度を検知することができる

図２　RealSense D435i

- 実験フィールド

　今回は実験のために以下のようなフィールドを作成した。この中に RealSense を置き、STL データから点群を生成することでマッチングする。

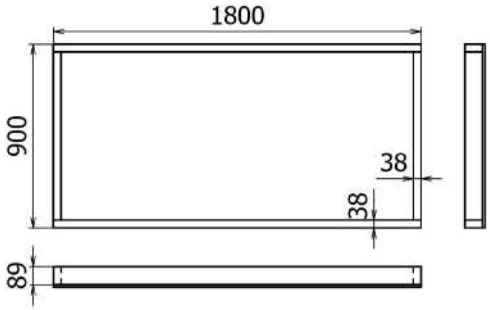

図３フィールド図面

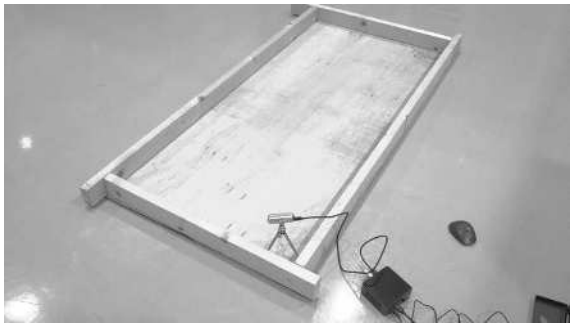

図４フィールド

手法

　自己位置推定はNDTマッチングと呼ばれる方法を用いて行う。この手法では地図データをボクセル（立方体に）分割し、ボクセルごとの点の集合を正規分布で近似する。スキャンデータ周辺のデータのみを扱うので計算量はスキャンデータの大きさによって決定される。今回は PointCloudLibrary（以下 PCL）に実装されている機能を利用するためこれ以上詳しく解説しない。PCL の NDT の項目の参考資料である Magnusson Martin 氏の論文を参考資料に含めているので、詳しくはそちらを参照していただきたい。

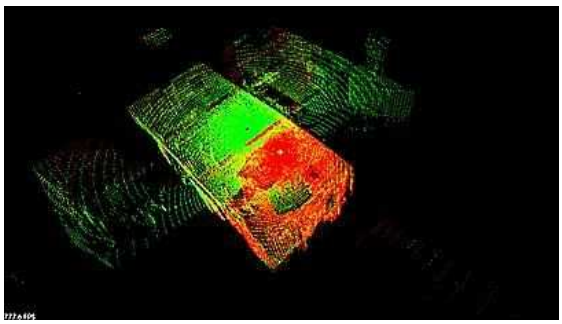

図５　NDT マッチングサンプル実行結果

手順

　マッチングの流れであるが、初めに RealSense から情報を取得する。RealSense の深度計測の解像度は 1280×720 であり、90 万個ほどの点が得られる。先述の通り、スキャンデータが少なければ少ないほど処理時間を短縮できるので PCL の VoxelGrid filter と呼ばれる機能で点群量を大幅に削減する。

　次に初期変換を定義する。スキャンデータをあらかじめ地図データに近い地点に設定することでマッチングを高速に行えるようにする。具体的には NDT マッチングで推定した位置を保存しておき、前回位置をあらかじめ読み込んでおくことで、一周期分の変位を推定すればよい。また IMU などを用いて加速度、角速度の積分をもって一周期分の変位を推定しておくことでより NDT によって推定

する量を減らすことができる。しかし今回は IMU による推定は研究期間の都合で実装はされていない。

最後に NDT マッチングを行う。NDT マッチングは複数回の反復を行うことで精度を高めることができ、評価値がしきい値を下回ると終了するようになっている。ここで推定された位置が自己位置となり、この値は次の位置推定に利用される。

結果

以下の図はマッチングの結果を表した図である。小さい点は RealSense がスキャンした点を表しており、大きな点が NDT によって座標変換されたあとのスキャンデータの位置である。また、きれいに整列している点はフィールドの枠である。RealSense はフィールドの角、写真では一番左側の角においてある。図からわかるように、地図データである木枠の沿って大きな点が分布しており、適切にマッチングされたことがわかる。また写真では伝えられないことが残念だが、RealSense を移動させてもマッチングし続けることも確認された。

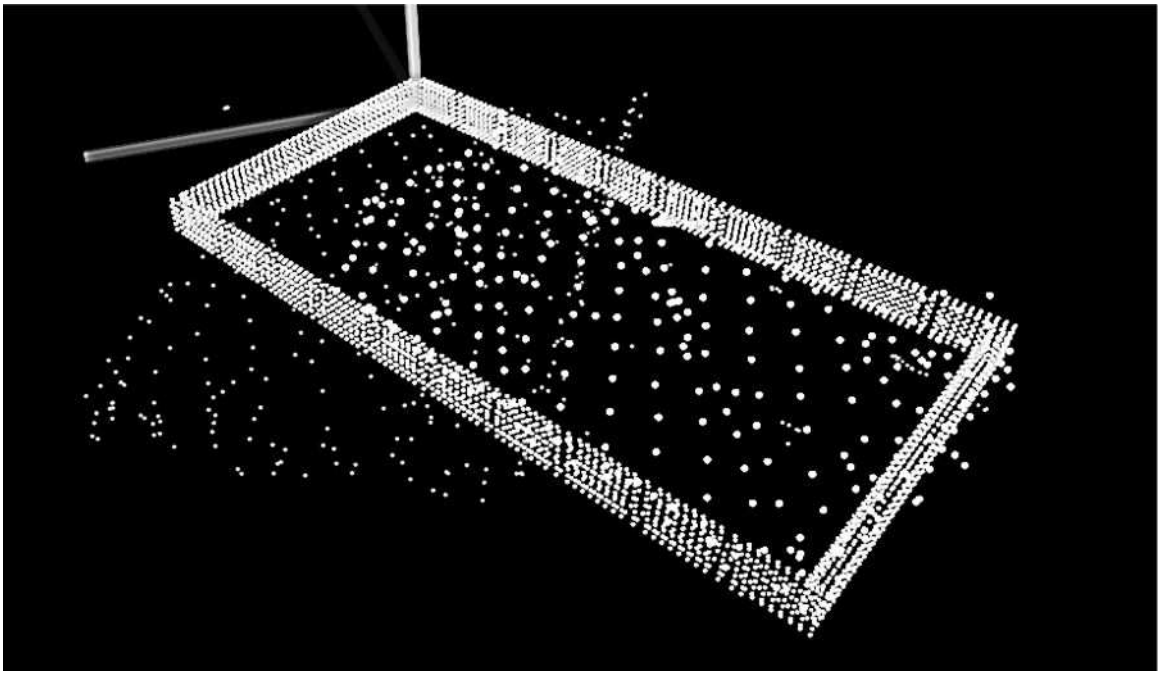

図６　マッチングの様子(ななめ視点)

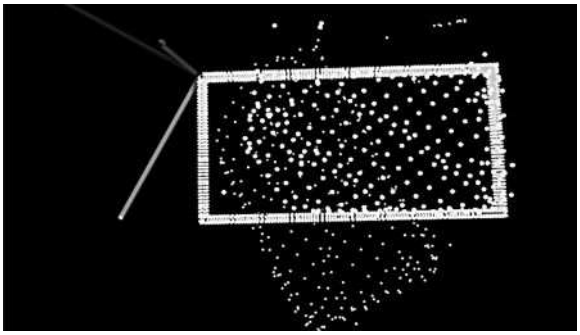

図７　マッチングの様子（真上視点）

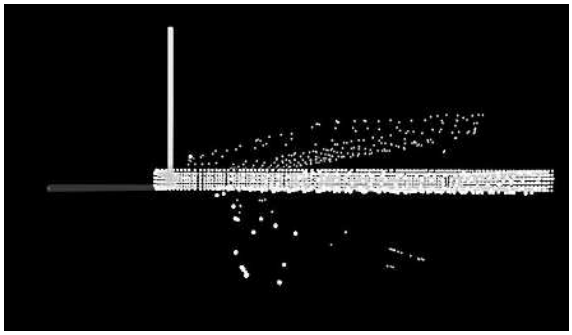

図８　マッチングの様子（真横視点）

しかし、問題点も多数ある。私の実装が不十分であるため処理が間に合っていない。というのも複数コアを用いたマルチスレッド処理にすることができなかったからだ。スキャンデータを 300 点ほど、反復回数を１回にしても、マッチング周期が 2Hz ほどしか出ていない。IMU による推定も行っ

ていないので、RealSense の移動速度が秒速 50 ㎜程度でなら利用できるがそれ以上では難しいという結果だった。しかし、裏を返せば IMU での推定のずれが秒速 50 ㎜以上の速さで広がらなければ可能であると思われるので、IMU による推定の実装が求められる。また、今回実験したのは単純な木枠内であるため、さらに複雑なフィールドで適応できるか検証する必要がある。

課題

本研究では期間内に行うことのできなかった内容がいくつか存在している。

1. マルチスレッド処理による高速マッチング。

   近年の計算機はシングルスレッドの速度の伸びは緩やかになってきており、その代わり多くのコア、スレッドを用いて計算することで性能を伸ばしている。本研究で使用した計算機も 4 コア 8 スレッドの製品であり、シングルスレッドよりマルチスレッドの方が 4 倍以上は速いということになる。挑戦はしたものの短期間で実現することはかなわなかった。

2. IMU を用いたマッチング補助

   NDT マッチングはマッチング対象との差が大きいほどマッチングに時間がかかる。IMU を用いることで検出される加速度角速度から現在位置を推定し、マッチング対象との差を減らすことでマッチング時間を短縮することができる。しかし、それには高周期で IMU の値を読み、積分をする必要性があるが、シングルスレッドだとマッチング速度が遅いため精度が出ないことが予想された。そのため実装には別スレッドを用意する必要があるが、NDT マッチングのマルチスレッド処理の開発を優先させたため挑戦すらできなかった。

3. 点群量減少プログラムの動作の理解

   RealSense の点群を減少させるために利用した PCL の機能が説明とは違うように感じられた。VoxelGrid filter と呼ばれるものを利用しているが、PCL の Documentation には「Then, in each voxel (i.e., 3D box), all the points present will be approximated (i.e., downsampled) with their centroid.」とあり、空間を区切り、その各々の中の点が一つの重心点に近似されると読める。しかし、実際は図９のような筋のようなものが入ってしまった。この原因は最後まで分からず、点群数が増加してしまい、処理を遅くしてしまった。

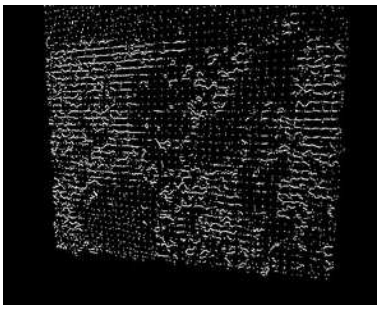

図９　縦筋が入った点群

4. 全方位移動台車

   予算不足で作成を断念した。ただ現状だと高速動作ができないため作成していても使われなかったであろう。

これらについては今後開発していければよいと考えている。

参考文献

[1] Point Cloud Library Documentation (閲覧日：2019/12/11)
http://pointclouds.org/documentation/

[2] Magnusson, Martin. (2009). The Three-Dimensional Normal-Distributions Transform --- an Efficient Representation for Registration, Surface Analysis, and Loop Detection.

[3] GitHub rs-pcl.cpp (閲覧日：2019/12/11)
https://github.com/IntelRealSense/librealsense/blob/master/wrappers/pcl/pcl/rs-pcl.cpp